# Symbolische manipulatie

Rik Voorhaar (3888169) - Jan-Willem van Ittersum (3992942) - Jurre Corver (3905985)
Begeleider: Joost Houben

June 29, 2015

# 1 Introductie

In deze opdracht hebben we een eigen computeralgebrasysteem (CAS) ontwikkeld om symbolisch te kunnen rekenen zoals dat bijvoorbeeld in Mathematica gebeurt. Wiskundige formules worden hiervoor opgeslagen in een zogenaamde expressie-boom. Behalve dat deze representatie kan worden gebruikt om berekeningen te doen, is deze geschikt voor symbolische manipulaties, zoals optellen, vermenigvuldigen, maar ook differentiëren en oplossen van sommige polynoom vergelijkingen. De code behordende bij dit project kan gevonden worden op `https://github.com/JurreCorver/SymbolischeManipulatie`.

# 2 Theorie

Een expressie-boom is een samenhangende, gerichte graaf, waarbij elke knoop (op één knoop na) exact één inkomende zijde heeft en een willekeurige aantal uitgaande zijden. De unieke knoop zonder inkomende zijde noemen we *de wortel*, een knoop met minstens één uitgaande zijde noemen we *een interne knoop* en een knoop zonder uitgaande zijde noemen we *een blad*. In de knopen van deze expressie-boom wordt een berekening opgeslagen. Een interne knopen (en de wortel) is een operator of een functie, een blad is een constante of een variabele. Een voorbeeld zie je in **figuur 1**

# 3 Algoritmen

# 4 Documentatie

## 4.1 Installatie

Om dit programma te gebruiken zijn naast een werkende Python 3.4 distributie enkele Python packages nodig. De niet-standaard packages zijn: `numpy, pillow, scipy, tkinter`. Verder heeft de grafische gebruikers omgeving onderstuining voor het weergeven van de output met behulp van LaTeX. Hiervoor is dus een werkende LaTeX distributie vereist. Daarbij is het ook benodigd om *GhostScript* geinstalleerd te hebben. Voor *Windows* gebruikers is het verder specifiek vereist om de 32-bit versie van *GhostScript* te gebruiken en het pad naar `gswin32c.exe` toe te voegen aan de `%path%` systeemvariabele.

## 4.2 Grafische omgeving

De grafische omgeving kan gebruikt worden door `tkmain.pyw` uit te voeren. Dit kan ook vanuit de command-line door `python tkmain.pyw` uit te voeren. De verschillende componenten van de omgeving zullen worden uitgelegd met referentie naar figuur 1

1. Met de `Use LaTeX` wordt gespecificeerd of de output door LaTeXwordt verwerkt voor dat deze wordt weergegeven. Indien deze optie niet geselecteerd is zal de output als plain-text worden weergegeven.

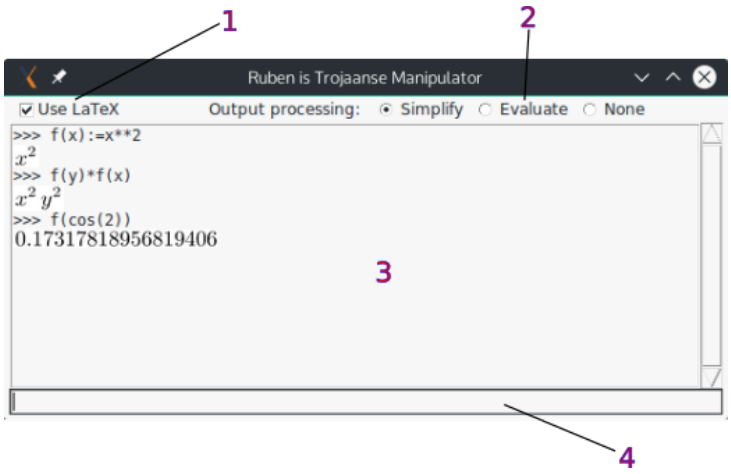


Figure 1: Screenshot van de grafische omgeving

2. De `Output processing` optie specificeerd hoe de input wordt verwerkt. Indien `Simplify` is aangevinkt zal de expressie zo veel mogelijk versimpeld worden. Terwijl bij het aanvinken van `Evaluate` de expressie wordt uitgerekend indien deze numeriek is, en anders onversimpeld wordt weergegeven. Als laatste zorgt de `None` optie juist dat de expressie niet wordt berekend en direct wordt weergegeven.

3. Dit is het scherm waar de output wordt weergegeven. Hier wordt de input van de gebruiker met `>>>` er voor weergegeven, en telkens op de regel daarna de output bij de regel input erboven. Indien er fouten optraden tijdens het uitvoeren van de input wordt dit hier ook weergegeven.

4. In dit scherm kan de gebruiker zijn input invoeren. Als de gebruiker de `Enter` toets indrukt wordt de input geëvaluaeerd. De pijltjestoetsen naar boven en onder kunnen worden gebruikt om de vorige input nog een keer te gebruiken.

## 4.3 Syntax

De invoer ondersteund standaard operaties op getallen die met symbolen `*, +, -, **, /, %` kunnen worden ingevoerd. Verder kunnen ronde haaken `( )` worden gebruikt. Enkele voorbeelden:

```
>>> 2+2-1
1
>>> (2**3)/2
4
>>> 7 % 3
1
```

Ook zijn standaard numerieke functies `sin, cos, tan, arcsin, arccos, ln, exp, floor, gamma, polygamma` geïmplementeerd. De specifieke details over deze functies zijn te vinden in de volgende sectie. Er is verder ook ondersteuning voor complexe getallen. De imaginaire constante wordt aangeduid met `i`. Andere standaard constanten zijn `pi, e, phi`. Voorbeeld:

```
>>> i**2
−1
>>> exp(i*pi)-1
0
```

Naast standaard operaties is het ook mogelijk om zelf functies en constanten toe te voegen. Hiervoor kan `:=` gebruikt worden. Als links van de `:=` alleen een symbool staat, dan wordt wat rechts van de `:=` staat opgeslagen onder naam van het symbool aan de linkerkant. We kunnen bijvoorbeeld het volgende doen:

```
>>> x:=5
5
>>> x**2
25
```

Functies kunnen op een vergelijkbare manier worden toegevoegd. Neem als voorbeeld:

```
>>> f(x,y):=sin(x)*cos(y)
sin(x) cos(y)
>>> f(pi/2,pi)
−1
```

Als laatste is er ondersteuning voor een gelijkheids operator met `==` als symbool. Deze kan bijvoorbeeld worden gebruikt bij invoeren van vergelijkingen.

## 4.4 Lijst van commando's

**na afloop op alfabetische volgorde zetten**

`exit()`
: Sluit het programma af.

`d(f(x),x)`
: Berekent de afgeleide van `f(x)` naar de variabele `x`. Indien de afgeleide van de functie niet bekend is wordt een foutmelding gegeven. Voorbeeld:

```
>>> d(sin(x)+2,x)
cos(x)
```

`sin(x)`
: Geeft de sinus van `x`.

`arcsin(x)`
: Geeft de arcsinus van `x`.

`cos(x)`
: Geeft de cosinus van `x`.

`arccos(x)`
: Geeft de arccosinus van `x`.

`tan(x)`
: Geeft de tangus van `x`.

`arctan(x)`
: Geeft de arctangus van `x`.

`log(x,y)`
: Geeft het logaritme van `x` in basis `y`. Equivalent aan `ln(x)/ln(y)`.

`ln(x)`

Geeft het natuurlijk logaritme van `x`.

`exp(x)`
Geeft het exponent van `x`.

`floor(x)`
Neemt de floor van `x`.

`gamma(x)`
Geeft de gamma functie van `x`.

`polygamma(x,y)`
Geeft de `y`'de orde polygamma functie van `x`.

**misschien ook nog even round en ceil toevoegen voor compleetheid?**

**nog toevoegen:**

`solvePolynomial(eq, var)`

`numIntegrate(expressie, x, l, r, numsteps)`
Numeriek bepalen van $\int_1^r$ `expressie` d`x` door middel van een Riemann benadering van `numsteps` stappen.

`gcd(n,m)`
Geeft de grootste gemene deler van twee gehele getallen `n` en `m`.

`polQuotient(pol1, pol2, x)`
Geeft het quotient van de deling van `pol1` door `pol2` met rest, waarbij `pol1` en `pol2` polynomen in `x` zijn.

`polRemainder(pol1, pol2, x)`
Geeft de rest van de deling van `pol1` door `pol2`, waarbij `pol1` en `pol2` polynomen in `x` zijn.

`polIntQuotient(pol1, pol2, x)`
Geeft het quotient van de deling van het polynoom `pol1` door het polynoom `pol2` met rest over de gehele getallen, waarbij `pol1` en `pol2` polynomen met gehele coëfficiënten in `x` zijn.

`polIntRemainder(pol1, pol2, x)`
Geeft de rest van de deling van `pol1` door `pol2` over de gehele getallen, waarbij `pol1` en `pol2` polynomen in `x` zijn.

`polContent(pol,x)`
Geeft de content van het polynoom `pol` in `x`.

`polGcd(pol1,pol2,x)`
Geeft de grootste gemene deler van twee polynomen `pol1` en `pol1` met gehele coëfficiënten in `x`

`deg(pol,x)`
Geeft de graad van `pol` als polynoom in `x`

# 5 Resultaten

# 6 Taakverdeling

# References

[1] Wikipedia - Shunting-yard algorithm: `https://en.wikipedia.org/wiki/Shunting-yard_algorithm`. - geraadpleegd op 28 juni 2015

[2] Houben, Joost. Expressie-bomen en Symbolische Manipulatie: WISB256 – Programmeren in de Wiskunde.